Project 1: Threads

Winter 2009

Jason Bau
Stanislas Polu

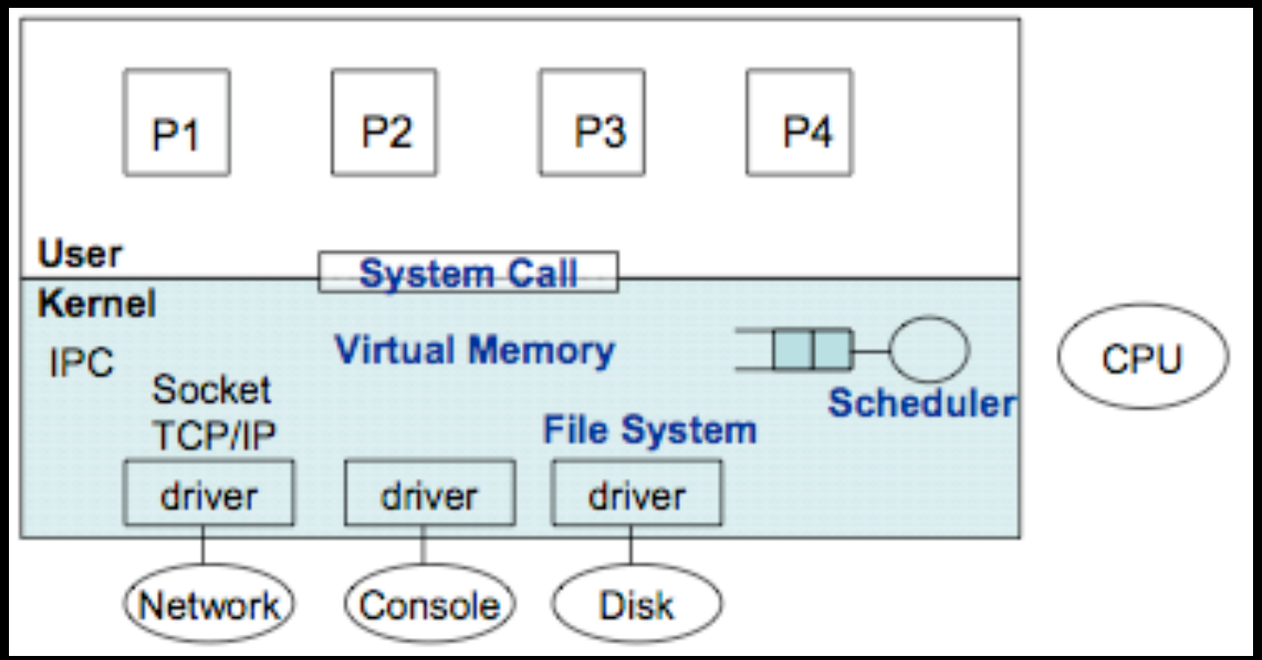
Based on slides from previous CA, Pr Mazières, Pr Rosemblum

Overview

- Threads Basics
- Project goals
  - Alarm Clock
  - Priority Scheduling
  - Advanced Scheduler (MLFQS)
- Getting Started

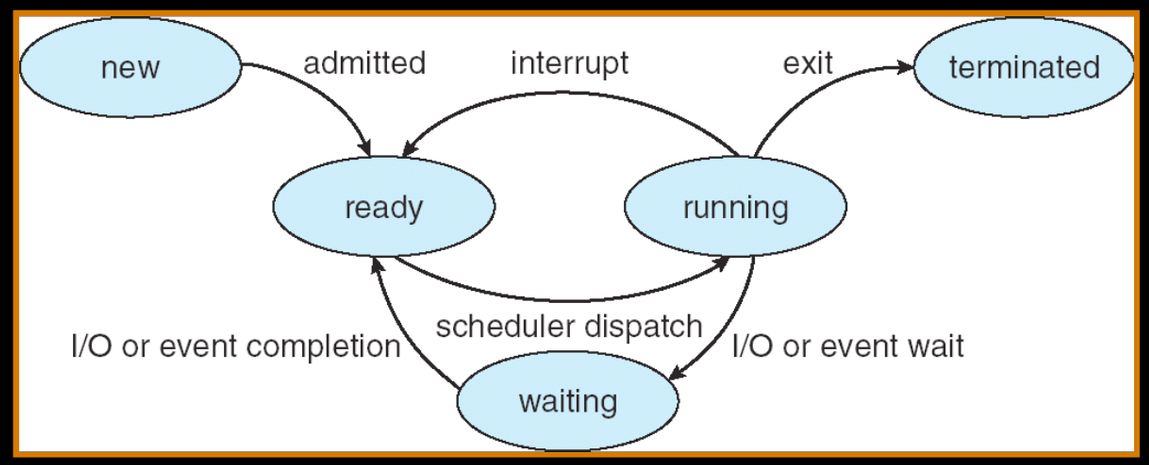
Basics

- OS Structure



Basics

- Thread ~ pointer to instruction & state
  “execution stream in an execution context”
- Key OS Aspects:
  - Maintain per-thread state
  - Pick a thread to run
  - Switch between threads



Basics

- Per thread state

```
typedef struct tcb {
  unsigned long md_esp;                 /* Stack pointer of thread */
  char *t_stack;                        /* Bottom of thread stack */
}
```

- Machine dependent thread switch / init

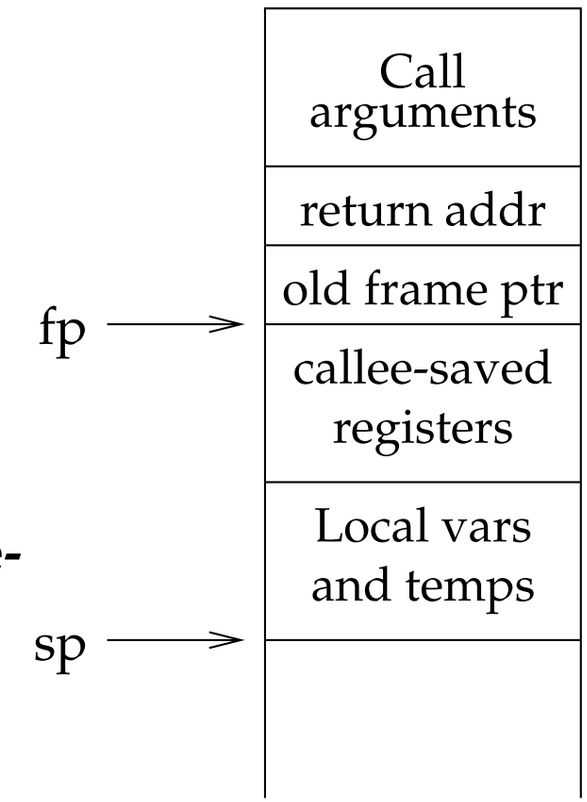
```
void thread_md_switch (tcb *current,  tcb *next)
```

```
void thread_md_init (tcb *t,  void (*fn) (void *),  void *arg)
```

Basics

Background: calling conventions

- ***sp* register always base of stack**
  - frame pointer (*fp*) is old *sp*
- **Local vars in stack & registers**
  - By convention, registers divided into caller- and callee-saved
- **Function arguments go in callee-saved regs and on stack**



Basics

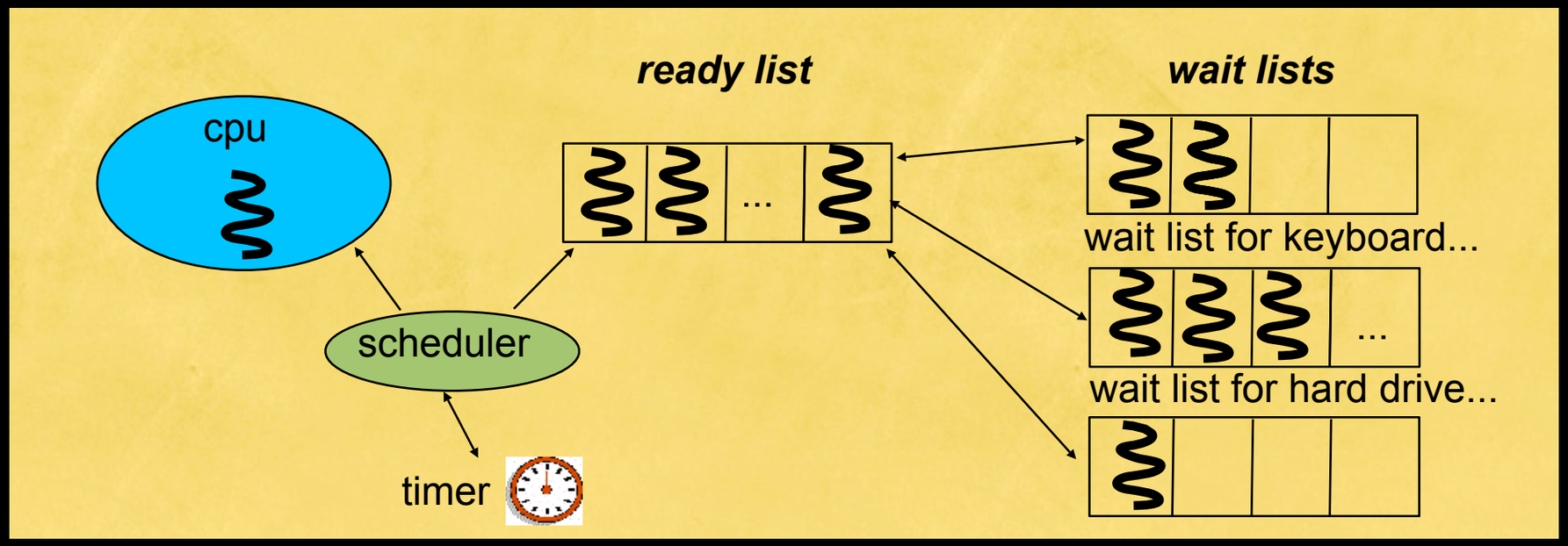
i386 thread_md_switch

```
pushl %ebp; movl %esp,%ebp           # Save frame pointer
pushl %ebx; pushl %esi; pushl %edi   # Save callee-saved regs

movl 8(%ebp),%edx        # %edx = thread_current
movl 12(%ebp),%eax       # %eax = thread_next
movl %esp,(%edx)         # %edx->md_esp = %esp
movl (%eax),%esp         # %esp = %eax->md_esp

popl %edi; popl %esi; popl %ebx     # Restore callee saved regs
popl %ebp                           # Restore frame pointer
ret                                 # Resume execution
```

- **This is literally switch code from simple thread lib**
  - Nothing magic happens here
- **You will see very similar code in Pintos switch.S**

Basics

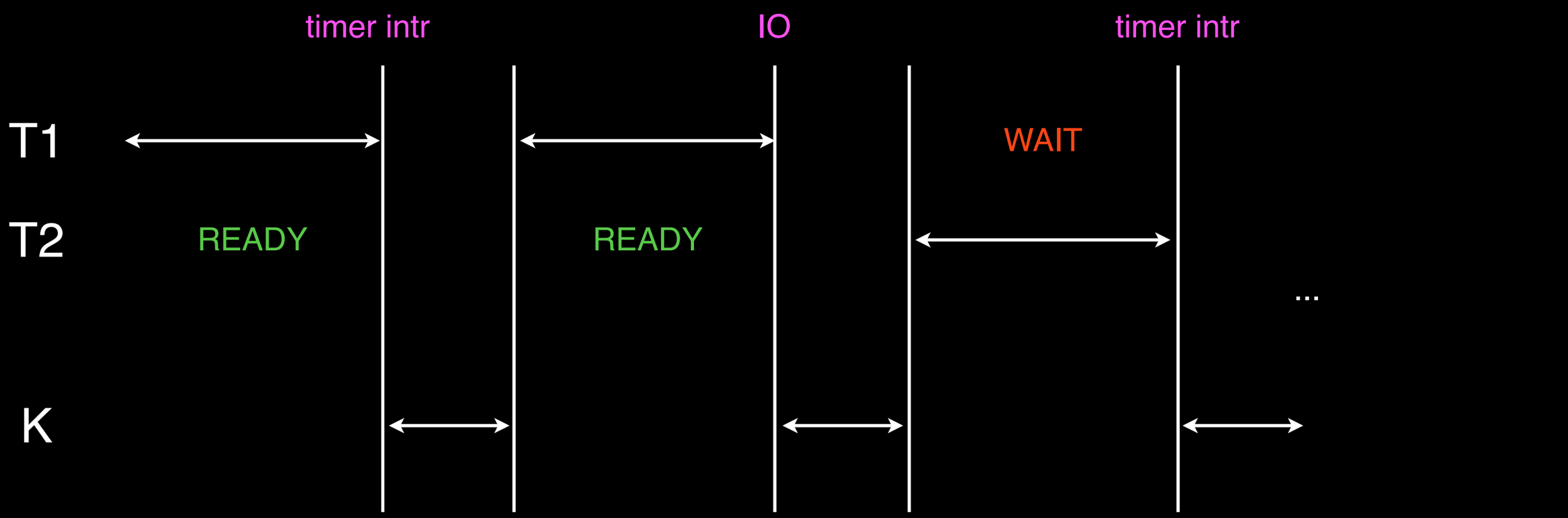
- Thread system overview




```
while (1)
{
    interrupt thread
    save state
    get next state
    load state, jump to it
}
```

Basics

- Context Switch



Project I

I. Alarm Clock

- Reimplement timer_sleep()
  - Avoid busy wait (why expensive?)
  - Instead take thread off the ready list (to where?)

```
## devices/timer.c

void timer_sleep (int64_t ticks)
{
     int64_t start = timer_ticks ();
     ASSERT (intr_get_level () == INTR_ON);

     while (timer_elapsed (start) < ticks)
          thread_yield ();
}
```

IIa. Priority Scheduling

- Priority Scheduling :
  - Thread L yields as H added to ready list
  - Thread H wakes up first when H and L both waiting for a lock, semaphore, or conditional variable.
- Needed before Part III

IIb. Priority Donation

- Priority Inversion Problem:
  - L holds lock K, running
    - H comes in ready list, kicking out L (L still holds K)
    - M comes in ready list
    - H waits for K, M starts running
  - Now M runs, then L, then H

IIb. Priority Donation

- Priority Donation:
  - Donate H priority to L
  - You must handle multiple donation to a same thread
  - You must handle nested donations H->M->L

  - Required for locks (sema, cond_vars optional)

III. Advanced Scheduler

- BSD Scheduler
  - Appendix B4.4
  - Priority depends on niceness, recent_cpu, load_avg

- Fixed-Point Real Arithmetic needed

Synchronization

- Threads can be interrupted anytime, use locks, semaphore and condition variables
- What happens when interrupts disabled?
- Can an interrupt handler hold a lock?

Grading

- 50% Design Document
  - Use Template and Example
- 50% Test Suite
  - run ‘make check’ in build/
  - Test scripts are in ‘pintos/src/tests’

Getting Started

- Make sure pintos is running
  - set path = (/usr/class/cs140/`uname -m`/bin $path)
  - tar xzf /usr/class/cs140/pintos/pintos.tar.gz
  - cd pintos/src/threads/
  - make
  - cd build/
  - pintos -v -k -- run alarm-multiple

Getting Started

- How to debug ?

```
vine1:~/pintos/src/threads/build> pintos -v --gdb -- run alarm-multiple

Writing command line to /tmp/nWbB7R3jwN.dsk...
squish-pty bochs -q
========================================================================
                Bochs x86 Emulator 2.2.6
        Build from CVS snapshot on January 29, 2006
========================================================================
00000000000i[     ] reading configuration from bochsrc.txt
00000000000i[     ] Enabled gdbstub
00000000000i[     ] installing nogui module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt

Waiting for gdb connection on localhost:1234
```

Then... from the *SAME* machine use:

```
pintos-gdb kernel.o
```

and issue the command:

```
target remote localhost:1234
```

Getting Started

- Example GDB Session

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234

(gdb) b thread_init
Breakpoint 1 at 0xc0101a65: file ../../threads/thread.c, line 114.

(gdb) c
Continuing.

Breakpoint 1, thread_init () at ../../threads/thread.c:114
114     {
(gdb)
```

Getting Started

- How to run the test suite?

```
vine1:~/pintos/src/threads> make check
```

- How to run an individual test?

```
vine1:~/pintos/src/threads> make build/tests/threads/alarm-multiple.result
```

```
vine1:~/pintos/src/threads/build> pintos -v -- run alarm-multiple
```

Useful Tools

- SCM
  - CVS / SVN / git
- Development tools
  - cscope, backtrace, pintos-gdb
- Data structures
  - especially lists ! (pintos/src/lib/kernel/)
- Newsgroup

Advices

- Read the manual
- Read the code
- Read the manual
- Read the code
- Read the manual
- Read the code
- ...

Advices

- Spend a LOT of time reading manual and code
- Work early on Design Document
- Integrate EARLY